Public Comments on Block Cipher Modes of Operation (Updated September 28, 2004)

This document contains the public comments on block cipher modes of operation that NIST received by electronic mail after the First Modes of Operation Workshop.

Commenter	**Date Received**	**Page**
David Scott	March 1, 2001	2
Paulo S. L. M. Barreto Scopus Tecnologia S.A.	April 30, 2001	4
Paulo S. L. M. Barreto Scopus Tecnologia S.A.	May 9, 2001	5
Paulo S. L. M. Barreto Scopus Tecnologia S.A.	May 30, 2001	6
William A. Simpson	August 13, 2001	7
Henrick Hellström	August 19, 2001	9
Greg Rose Qualcomm Australia	August 29, 2001	10
Paulo S. L. M. Barreto Scopus Tecnologia S.A.	August 29, 2001	12
Jesse Walker Intel Corproation	August 31, 2001	15
Tony Jeffree IEEE 802 LAN/MAM Standards Committee IEEE 802.1 working group	March 24, 2004	16

David Scott

Dear Sir
below is text on RFC1423
"
http://www-users.aston.ac.uk/Connected/RFC/1423/rfc1423.txt

```
Balenson                                                  [Page 2]

RFC 1423        PEM: Algorithms, Modes and Identifiers    February 1993

   DES is defined in FIPS PUB 81 [3], and is equivalent to those
   provided in ANSI X3.106 [4] and in ISO IS 8372 [5].  The character
   string "DES-CBC" within an encapsulated PEM header field indicates
   the use of this algorithm/mode combination.

   The input to the DES CBC encryption process shall be padded to a
   multiple of 8 octets, in the following manner.  Let n be the length
   in octets of the input.  Pad the input by appending 8-(n mod 8)
   octets to the end of the message, each having the value 8-(n mod 8),
   the number of octets being added.  In hexadecimal, the possible
   paddings are:  01, 0202, 030303, 04040404, 0505050505, 060606060606,
   07070707070707, and 0808080808080808.  All input is padded with 1 to
   8 octets to produce a multiple of 8 octets in length.  The padding
   can be removed unambiguously after decryption.
```
"

The above is similar to the kind of padding we may be stuck with
for the new AES. It has some major weaknesses that are easy to fix.
The main weakness is that allows an attacker to throw away many
keys when trying to break a file since most keys will not lead
to a file of the above form when using the wrong key for decyrption.

I would like to propose a better solution that does not weaken
the encryption like the current standard method. I would be willing
to present this type of padding if needed. Since I am a retired
GS-13 engineer. My government service was at China Lake.

The concept is very simple. And involves a few very simple steps.
1) the starting file is an arbitrary 8-bit byte type of file.
Convert it to an infinite file that is "finitely odd".
An easy method of doing this is to look at each 8 bit byte in the
file. If the trailing bytes are not (all values in HEX) "00"
or "00" followed by one or more "80" then leave file alone
and pretend it is followed by an infinite number of "zeroes".
If the file does have a trailing "00" or "00" followed by
one or more "80" then add a trailing byte of "80" and then
follow that by an infinte number of zeros.
Examples:
00 00 becomes 0000800000000000...
80 becomes 800000000...
00 80 becomes 0080800000000000...
30 80 becomes 3080000000000000...
40 00 becomes 4000800000000000...
23 12 becomes 2312000000000000...

note all files at this point have a last one "bit" followed
by a tail of an infinite number of zeros.

2) convert this to a normal file the block size of the
encryption. Lets assume the 16 byte block of the AES method.
To convert group the infinte finitely odd file into blocks
of 128 bits. Then ignore the infinite number of zero blocks
that trail at end. At this point the last block can not be
all zeros. So the next step is to look at the last block
remaining. If it is of form "80 00 00 rest of block zero "
at this poiint if last block is this check the next to last
block until either you reach the start of file or a block that
is not of form "80 00 00 rest of block zero" If the block
your at is a totally "zero block" you drop the last block. If
not you dont.
Examples here "zz" means rest of 16 byte block zero:

80 00 zz 80 00 zz 0000000000000000000...
 becomes 80 00 zz 80 00 zz
00 00 zz 80 00 zz 0000000000000000000...
 becomes 00 00 zz
23 00 zz 78 00 zz 0000000000000000000...
 becomes 23 00 zz 78 00 zzzz

3) since all files have been "1 to 1" mapped to the desired
block size do the encryption.

4) Now the files are encrypted and in the correct block
size. At this point convert them back to finitely odd files.
This is like step 1 but instead of 8 bit bytes you use 128 bit
chunks. The token equaivalent to "00" is the block of 128 bits
all zero. And the one equivalent to "80" is the block of 128 bits
all zero except the first bit (left most) is a one.

5) convert this file doing the exact opposite of step one.
This gives the final encrypted output file.

ONE SPECIAL PROPERTY of this kind of padding. Is
that any 8-bit binary file can be thought of as an
output file to the AES encryption. And encryption
key test for any binary file. Will lead to a decrypted
file that when encrypted with the method above goes back
to the same file.
 I am willing to give a presentation on this if more
explanation is needed.

Thank You
David A. Scott

Paulo Sérgio Licciardi Messeder Barreto

Dear NIST AES team:

Integrity-aware modes of operation (i.e. modes whose goal is to provide not only confidentiality but integrity and authentication as well) have recently received a lot of attention from researchers. Some proposed modes, particularly the Offset Codebook (OCB) mode by Rogaway et al., clearly indicate how quickly the concept matured, both from the viewpoint of security analysis and that of engineering considerations. For developers and other direct users of this technology, it became undeniable that integrity-aware modes present concrete and substantial advantage over conventional modes: in many scenarios, the benefits gained far outweigh the inconvenient of patents.
Therefore, it would be a loss if NIST would choose not to include one such mode for use with the AES.

Some months ago I expressed the opinion that NIST should only standardize conventional, unpatented modes. That opinion obviously could not take into account the analyses now available for some of the new modes. The OCB submission document in particular is one of those rare pieces of research that manage to bind together thorough analysis (to my knowledge, the deepest among the new mode proposals), very clear and complete explanation, ease of use, and clean reference code.

Hence, I hereby withdraw my initial opinion, and strongly recommend standardization of OCB together with the more conventional modes.

Best regards,

Paulo S. L. M. Barreto.
Chief Cryptographer.
Scopus Tecnologia S.A.

Paulo Sérgio Licciardi Messeder Barreto

Dear NIST team,

On Thu, 01 Mar 2001, David Scott <dscott@elpasonet.net> submitted comments to NIST ("method of padding for arbitrary length files") asserting that the padding method described in RFC 1423 "has some major weaknesses", the main one being that it "allows an attacker to throw away many keys when trying to break a file since most keys will not lead to a file of the above form when using the wrong key for decyrption" [sic].

I would like to point out that this behavior can hardly be considered a "weakness". The above approach to recover the key is nothing more than exhaustive key search (with a fragment of known plaintext), whose expected computational workload is by no means decreased by the RFC 1423 padding.

The RFC 1423 padding does introduce a redundancy that is exploitable as a known plaintext (assuming the attacker knows the original data length); however, any other known property of the plaintext itself (not explicitly the padding) could be used instead.

I think the true limitation of the RFC 1423 padding is that it makes no provision for padding data which is not octet-aligned (i.e. whose bit length is not a multiple of 8). Alternative choices might be:

1. Set all padding octets but the last to random values, and set the last octet to the number of padding *bits* (as opposed to bytes). This, however, still has some drawbacks (for instance, it demands the availability of a PRNG).

2. Use a hash-like padding: append a 1-bit and then as many 0-bits as needed to make the length of the resulting bit string a multiple of 128 (the AES block size). This approach is perhaps the simplest and the most general.

This, of course, only applies to modes of operation that need padding at all, and provides an interesting argument in favor of modes that don't need that kind of construction, like CTR or OCB.

Best regards,

Paulo S. L. M. Barreto.

Paulo Sérgio Licciardi Messeder Barreto

Dear NIST team,

Since the AES-Hash mode of operation was proposed long after the submission deadline, and nevertheless is the only proposal that addresses a *hash* mode, I would like to suggest that discussion of this mode (and/or other hash modes for the AES) be made together with, or parallel to, the discussion of the SHA-2 family of hash functions.

Ths motivation for this suggestion is threefold:

1. Keeping the current NIST deadlines for the AES FIPS and the Modes of Operation FIPS, while providing additional time (namely, the lapse until the SHS deadline) for deeper scrutiny of hash modes;

2. Avoiding mixing the discussion of encryption modes with that of hash modes, which are likely to raise quite different issues, requirements, and analyses;

3. Making it possible, if deemed of interest, to include AES hash modes in the revised secure hash standard document (thus centralizing in one document the overall subject of cryptographic hashing, in the same way the latest version of the DSS centralizes the overall subject of digital signatures for all adopted public key algorithms).

4. Keeping open the possibility of offering an alternative to the SHA-2 functions for constrained environments (like smart cards and dedicated hardware) where the AES is already implemented.

Best regards,

Paulo S. L. M. Barreto
Chief Cryptographer
Scopus Tecnologia S. A.

William Allen Simpson

I was recently directed to the IACBC-IP.PDF claims. Although I cannot attend your conference (due to a conflict with another conference beginning the next day), please consider these brief comments.

--
I support the inclusion of an Integrity-Aware CBC mode, but would prefer that it be called Cipher Block CheckSum (CBCS), as it has been in my original messages and papers circa 1994-1997.

Although these papers were directed at the specific application of Internet Protocol Security (IPSec), of which I was a principle author, the CBCS modes are sufficiently general to use in any AES application.

CBC with checksums was first discussed on the IPSec mailing list in 1994, and formally presented at IETF. Specifications were distributed world-wide via the IETF on-line collection of internet-drafts.

Whitening the plaintext and/or ciphertext with a constant, pseudo-random sequence, cryptographic hash used as a pseudo-random function, and such combinations were documented in part 4 of "Intemet Security Transform Enhancements", draft-simpson-ipsec-enhancement-00.txt, April 1996.

The more recent "Cipher Block CheckSums (CBCS)" was first documented in draft-simpson-cbcs-00.txt, July 1997, although earlier less generic versions were discussed on various mailing lists.

These capabilities have all been implemented in the "Photuris: Session-Key Management Protocol" (RFCs 2522 and 2523), and discussed on the Photuris implementors' mailing list.

--
The IACBC diagrams are subsumed in the CBCS diagrams.

Variants with only plaintext whitening (series Sa) or only ciphertext whitening (Sb) are described in the text.

CBCS2 with 1 encryption key (k) and 2 integrity keys (Sa, Sb):

```
      IVa   P1          P2          Pi
       |     |           |           |
     +-----+  v  +-----+    v  +-----+    v
Sa->|  S  |->(X)->|  S  |->->(X)->|  S  |->->(X)->
     +-----+  |  +-----+    |  +-----+    |
             v     ^       v     ^       v
          +-----+  ^    +-----+  ^    +-----+
      k->|  E  |  ^  k->|  E  |  ^  k->|  E  |
          +-----+  ^    +-----+  ^    +-----+
             |     ^       |     ^       |
      IVb    +->->->+      +->->->+      +->->->
       |     |           |           |
     +-----+  v  +-----+    v  +-----+    v
Sb->|  S  |->(X)->|  S  |->->(X)->|  S  |->->(X)->
     +-----+  |  +-----+    |  +-----+    |
             v   ^    v    ^    v
             +->->->+    +->->->+    +->->->
             |       |        |
             C1        C2       Ci
```

--
The IBM patent may be restricted to their particular summation function
Fk(), several of which are suggested for various block sizes.

CBCS and CBCS2 both recommended a different function, specified
generically, independent of block size:

1) addition modulo 2**N with end around carry (N = bits in block);
2) count of the number of 1-bits in the sum (population count);
3) left circular rotation of the sum by the bit count.

Any such F() could be substituted, and the function specification is
not a requirement of CBCS generically. As a practical matter, a single
function MUST be chosen for interoperability.
--
William Allen Simpson
    Key fingerprint = 17 40 5E 67 15 6F 31 26 DD 0D B9 9B 6A 15 2C 32

Henrick Hellström

I regret that we are not able to present PCFB mode ourselves at the modes workshop. We do believe in the need for a secure, efficient, easy to implement and compact integrity aware encryption mode. At present we would advocate ABC mode combined with the AREA authentication scheme found in our proposal. We fear that the existential forgery attacks against DCTR mode found in the recent paper by Donescu, Gligor and Wagner, in some settings could be extended to the XECB and IAPM modes. If we were to choose between an error propagating integrity aware mode and a seekable integrity aware mode, the findnings of Donescu, Gligor and Wagner would presently definitely make us choose the former, all other things alike.

Furthermore, we wish to point out that the security bounds of PCFB are fully satisfactory. The security bounds w.r.t. secrecy makes it a high security mode - PCFB-64/128 is equivalent to using two interleaved key streams generated by block ciphers with a 64 bit block size, where the first keystream is xor:ed with the message text and used to continuously seed the second, and the second is kept secret and used to continuously seed the first. The security bounds w.r.t. integrity for PCFB-64/128 are 2**64, and this bound is entailed by the birthday paradox and the 128 bit block size - no other mode can do better against _any_ attack.

Finally, we have noted that PCFB-mode could be slightly adjusted without any security implications:

```
Adjusted PCFB-m/n, encryption
Input: n-bit vector V, m-bit plain text P
Output: n-bit vector V, m-bit cipher text C
1. T <- E_k(V)
2. C <- P xor (T mod 2**m)
3. V <- C or ((T>>m)<<(n-m))
```

The change is in step 3. In practice, this eliminates the need for any shift operations on the V / T variable. You simply replace the lower m bits of T with C, making the new value of V. There might be reasons not doing this with general CFB mode (since you don't need the V variable if you keep the cipher text), but we can't see any reasons not doing it with PCFB mode.

Regards,

Henrick Hellström

P.S. We will answer any further questions, and always appreciate comments, should there be any. D.S.

Greg Rose

At last week's NIST "second workshop on Modes of Operation" a number of participants expressed a desire for a combined mechanism which allowed authentication of an entire text, but only encrypted a part of it. One example demonstrating such a requirement is IPSec, where authentication and integrity is desirable for the entire packet including IP addresses but encryption is mostly meant for the payload.

We've worked out a way to do this, based on Jutla's submission "Parallelizable Encryption Mode with Almost Free Message Integrity", and we'll be using his notation below, but I do believe that what we describe below can be applied with only minor modification to Rogaway's and Gligor & Donescu's one pass methods.

Basic idea:

Looking at Jutla's Figure 1, it is quite obvious that there's great symmetry in the construct, that going forward (encryption) is similar to going backward, except for one thing: the checksum includes only the plaintexts. If the checksum is formed by including also the ciphertexts, that is,
  checksum = P_1 + C_1 + P_2 + C_2 ... + P_m-1 + C_m-1
(where '+' is the appropriate definition for the variants of the algorithm), nice things happen.

Now, by symmetry, it is just as hard to calculate/verify the checksum whether it is the Plaintext or the Ciphertext that is given. That is, the tag C_m is a valid MAC even when the message is sent clear. More importantly, the message can be sent partially clear, and partly encrypted, so long as whole blocks are either one or the other (and assuming agreement between sender and receiver as to which are which).

So the message sent might be {C_0, P_1, P_2, C_3, ... C_m-2, P_m-1, C_m) remembering that C_0 is the IV and C_m, the tag, must always be encrypted.

The cost of this construction is zero in terms of block cipher units (that is, assuming that the block XOR/ADDs are negligible). However in the completely parallelized implementation it does introduce one block encryption's worth of latency... the results of the encryptions of the message blocks must be known before the tag can be computed.

Reduced latency variant:

It might be possible to avoid this latency by further exploiting the symmetry argument by calculating *two* checksums -- the Plaintext checksum (P_c) and the Ciphertext checksum (C_c), and calculating the tag as (cf. code just before figure 1):

  M_m = (P_c + S_m) mod 2^128
  N_m = f_K1(M_m)
  C_m = (N_m + S_0 + C_c) mod 2^128

again varying meanings of "+" appropriately for the differing schemes.

We believe that it is self-evident that the former proposal (combined checksum before encryption of tag) is secure. I'm not convinced about this optimisation and would welcome someone with better formal proof skills to look at it.

Noting that information disclosed to the attacker can't particularly help a MAC to be secure, you might form the checksums by adding only those blocks which were not transmitted to the appropriate checksum. That is, for the example message shown above:

  C_c = C_1 + C_2 + C_m-1

P_c = P_3 + ... + P_m-2

This has identical performance to the original scheme except for one more block add/xor at the end.

What about partial blocks?

Clearly what we've written above relies upon entire blocks being either plaintext or ciphertext, which is an undesirable reflection of the structure of the block cipher. In particular, for IPSec, the parts that would be sent "clear" simply don't add up nicely to 128-bit blocks. We call a block a "partial block" when some of its contents are to be sent encrypted, and some of it is to be sent clear.

One way to solve this problem is to treat the partial block as a plaintext block for the purposes of calculating the various checksums above. Since this ensures message integrity, there should be no problem with encrypting the desired part with an additive cipher, we just need the right inputs to create it. The sequence of S_i values is assumed unknown to the attacker, as is the key, so encrypting the S_i should be an adequate source of keystream.

So, look at the 'i'th block, and assume that we have a bitwise mask M_i that defines the bits to be encrypted. (M_i won't be all zeros, or you'd just send the plaintext; similarly it won't be all ones, or you'd just send the ciphertext. But we don't need (or want) to constrain it to have leading zeros or anything like that... it's just a mask.) The block to be transmitted is calculated so:

Partial_i = P_i ^ (M_i AND f_K1(S_i))

As before, it is assumed that both parties know a priori the applicable mask M_i, as they already needed to know which blocks were sent as P_i/C_i.

The original block encryption is *still* needed to calculate C_i, for the checksum calculation, so this scheme costs another encryption operation per partial block, although it's still fully parallelizable.

Other considerations:

It's a minor point, but note that the recipient of a message in all cases above must be able and ready to do both decryption and encryption operations to check the tag and recover the plaintext. This was already somewhat true, as he needed encryption to create the tag sequence from the IV, but decryption otherwise, so we don't think it is a big drawback.

Idea by ("we") Phil Hawkes and Greg Rose
Written by ("I") Greg Rose

Greg.

PS to Morris Dworkin: you have my permission to mail this to the attendees of the workshop (or more widely) if you think that's a reasonable thing to do. And thanks to all involved for an enlightening and productive day.

Greg Rose INTERNET: ggr@qualcomm.com
Qualcomm Australia VOICE: +61-2-9817 4188 FAX: +61-2-9817 5199
Level 3, 230 Victoria Road, http://people.qualcomm.com/ggr/
Gladesville NSW 2111 232B EC8F 44C6 C853 D68F E107 E6BF CD2F 1081 A37C

Paulo Sergio Licciardi Messeder Barreto

Dear NIST,

Some brief comments on the AES-Hash proposal by Bram Cohen and Ben Laurie.

1. The choice of the compression function.

AES-Hash is a variant of the Davies-Meyer (DM) construction using Rijndael (with 256-bit block and key size) as underlying block cipher. The rationale for this choice is that DM:

(A) is one of the still-unbroken hash constructions, assuming certain properties of the underlying block cipher;
(B) enables computation of the key schedule used in one DM step to be done in parallel with the encryption in the previous DM step.

Although statement (A) corresponds to public knowledge [see e.g. A.J.Menezes, et al., "Handbook of Applied Cryptography", remark 9.45], statement (B) does not necessarily imply any advantage of DM over the remaining unbroken constructions, e.g. Matyas-Meyer-Oseas, at least for Rijndael. The reason is that, due to the lightweight Rijndael key scheduling algorithm, one can compute the round subkeys of any given DM step on the fly, in parallel with the very encryption step where they are used. More precisely, let k_i be the i-th Rijndael subkey corresponding to a given 256-bit cipher key k; k_0 equals k itself and is thus readily available, and for i > 0 one can compute k_i from k_{i-1} while evaluating the ByteSub, ShiftRows, and MixColumns transforms of the i-th round and the AddRoundKey transform of the previous round.

This setting has an added benefit for hardware and other storage-constrained environments, in that it is not necessary to precompute and store the whole key schedule of the next hashing step during the current step. Such precomputation would only be useful if the same key were used many times, as in conventional encryption modes. In a hashing mode, each key will (or is most likely to) be used only once. Therefore, any other unbroken scheme might be used instead of DM. This is not to say there is anything wrong with DM; it only means the current rationale is not enough to single out DM from among the other hashing scheme).

2. Avoiding continuation attacks.

Besides conventional Merkle-Damgaard (MD) strengthtening, the AES-Hash authors propose (eqs. (3) and (4) of the AES-Hash proposal) to apply an extra hashing step to the DM output to thwart continuation attacks -- that is, to prevent an attacker from being able, given the hash of some unknown data string S, to compute the hash of a longer string consisting of S concatenated with arbitrary data. However, such a mechanism is actually redundant: the usual message padding plus MD strengthening are enough for that purpose, unless one assumes that the attacker is interested in computing the hash of not entirely arbitrary extensions of S (e.g. extensions of S that begin with the very padding and length) -- in which case the proposed variant is as ineffective as MD to thwart a continuation attack, and needlessly increases the processing time. Hence it seems that the simpler, conventional DM contruction (with padding and MD strengthening) constitutes a more desirable setting.

3. Hashing constructions based on the AES parameters of Rijndael.

NIST has manifested its decision not to adopt any modes of operations that, like AES-Hash, employ Rijndael with block and/or key sizes other than those strictly compliant with the AES standard. As it turns out, it is still possible to define hashing modes of operation that do conform to the AES dimensions while producing longer (256-bit) digests, the two obvious candidates being Tandem and Abreast Davies-Meyer designed by X. Lai and J. Massey.

It seems that Abreast DM is the more advantageous of these two constructions, since the two AES encryptions involved (together with their key schedules) can be computed in parallel. Notice that the nature of parallelism here is different from the one considered in section 1, because there is no way to compute in parallel the two encryptions needed by Tandem DM.

A clear drawback of these constructions is that the hash rate is at most half that achievable by AES-Hash and similar modes, due to the unfavorable block size ratio. However, a hashing mode for AES would be normally intended for constrained environments that lack a dedicated hashing function like SHA-256. Under such circumstances, the tradeoff between hashing speed and code/storage size may be quite acceptable, or perhaps even unavoidable, in face of the already existing limitations.

It has been pointed out [D. Wagner, coderpunks discussion list posting, 2001] that the security of both Tandem DM and Abreast DM depends heavily on the key schedule of the underlying block cipher, and that Rijndael has not been subjected to scrutiny in this context to the same extent it has been regarding conventional encryption modes. On the other hand, one can argue that, first, no concrete attack (not even an "academic" attack) is known against Rijndael in these hashing modes, and second, these modes were almost certainly designed with the IDEA block cipher in mind, and though the IDEA key schedule is known to have weaknesses (namely, the weak keys discovered by J. Daemen) not found in Rijndael, it seems nevertheless suitable for use with either of the hashing modes designed by Lai and Massey -- put another way, these hashing modes appear robust even though they are used with a block cipher whose key schedule is no stronger (and perhaps even weaker) than Rijndael's.

4. Conclusion.

The topics discussed above may be summarized as:

(1) Any of the unbroken compression function constructions could be used. The AES-Hash rationale offered by the authors for the choice of DM is actually irrelevant; it would be much easier to justify the choice on the basis of the simplicity of DM.

(2) In a hashing mode of operation for AES, MD-strengthtening is sufficient to thwart continuation attacks. A construction like the mechanism suggested in the AES-Hash proposal is redundant at least, and ineffective in any case.

(3) If a hashing mode of operation is desired that produces 256-bit hashes

within the strictly AES parameters of Rijndael, Abreast DM is an attractive
possibility - contrary to the constructions where the key and block size are
the same, there seem to be only two obvious choices here, and Abreast DM is
arguably more advantageous from the point of view of exploitable
parallelism.

Best regards,

Paulo S. L. M. Barreto
Chief Cryptographer
Scopus Tecnologia S. A.

Jesse Walker

Sirs,

Please consider the adoption of an authenticated encryption mode of operation for AES.

I believe this type of mode of operation has the potential to save substantial costs for customers of commercial communication equipment that include embedded cyprotgraphic protocols. In particular, designs based on the classical modes of operations tend to break down on cost grounds at about 10 Gbps, and we are already designing network equipment to run at much higher rates that this. The silicon cost for crypto acceleration hardware for a 100 Mbps medium is only pennies, but rises sjarply to about the same cost as an entire NIC at 1 Gbps; the same types of designs starts fo become cost prohibitive above OC-48 rates.

An authenticate encryption mode also is easier to use correctly than separate encryption and data authenticity algorithms, so is more likely to result in correct protocol design by individuals with little cryptographic expertise. This is an important quality, as more and more media designs are appearing without proper review by cryptographers; 802.11, HomeRF, Bluetooth, HyperLAN II, and HomePlug all come to mind as having inadequate schemes that could have been easily avoided had proper tools like authenticated encryption been available to novice security protocol designers.

I particularly recommend your consideration of Offset Codebook mode, as it seems better engineered for real-life application than the other excellent submissions in this category.

Jesse Walker
Intel Corproation
JF3-448
2111 NE 25th Avenue
Hillsboro, OR 97214-5961
(503) 712-1849
jesse.walker@intel.com

Tony Jeffree

At the March 2004 Plenary meeting of the IEEE 802 LAN/MAM Standards Committee, the IEEE 802.1 working group agreed to forward the following liaison statement and request to NIST regarding Modes of Operation for Symmetric Key Block Ciphers:

"The 802.1 Working Group intends to use GCM-AES (Galois Counter Mode) as the mandatory cipher suite for the proposed IEEE P802.1ae Media Access Control Security standard.

At present the characteristics of GCM make it the only AES mode that would meet all our requirements.

The work of 802.1 would be greatly facilitated by the adoption of GCM by the NIST modes process."

Regards,
Tony Jeffree
Chair, IEEE 802.1 Working Group.